

# Asynchronous Gibbs Sampling

Alexander Terenin

Applied Mathematics and Statistics, University of California, Santa Cruz

aterenin@ucsc.edu

Dan Simpson

University of Bath (UK)

d.simpson@bath.ac.uk

David Draper

Applied Mathematics and Statistics, University of California, Santa Cruz

draper@ucsc.edu

**Abstract.** Gibbs sampling is a widely used Markov Chain Monte Carlo (MCMC) method for numerically approximating integrals of interest in Bayesian statistics and other mathematical sciences. It is widely believed that MCMC methods do not extend readily to parallel implementations needed in big data settings, as their inherently sequential nature incurs a large synchronization cost. In this paper, we present a novel scheme – *Asynchronous Gibbs sampling* – that allows us to perform MCMC in a parallel fashion with no synchronization or locking, avoiding the typical performance bottlenecks of parallel algorithms. Our method is especially attractive in settings, such as hierarchical random-effects modeling in which each observation has its own random effect, where the problem dimension grows with the sample size. We present two variants: an exact algorithm, and an approximate algorithm that does not require transmitting data. We prove convergence of the exact algorithm. We provide examples that illustrate some of the algorithm’s properties with respect to scaling, and an example that compares the exact and approximate algorithms.

**Keywords:** Bayesian mixed-effects models, Big Data, fixed-point algorithm, Gaussian Process regression, high-dimensional statistical modeling, Markov Chain Monte Carlo, parallel computing, synchronization.

## 1 Introduction

The Bayesian statistical paradigm has well-known optimality properties: given a full model specification through a prior and likelihood, a theorem due to Cox [6] (under very broad assumptions – see Terenin and Draper [36]) says that inferential uncertainty can be characterized in one and only one way through the use of conditional probability via Bayes’ Rule. Additionally, the paradigm offers extensive modeling flexibility, and an immediate way of determining what a statistical model assumes and what it can learn from the data.

The e-commerce company eBay, Inc. is interested in using the Bayesian paradigm for purposes of inference and decision-making, and employs a number of Bayesian models as part of its operations. One of the main practical challenges in using the Bayesian paradigm on a modern industrial scale is that the standard approach to Bayesian computation for the past 25 years – Markov Chain Monte Carlo (MCMC) [15, 22] – does not scale well, either with data set size or with model complexity.

This is especially of concern in e-commerce applications, where typical data set sizes range from  $n = 10^6$  to  $n = 10^{10}$  observations.

In this paper we offer a new algorithm – *Asynchronous Gibbs sampling* – that removes synchronicity barriers which hamper the efficient implementation of most MCMC methods in a distributed environment. This is a crucial element in improving the behavior of MCMC samplers for complex models that fall within the current “Big Data/Big Models” paradigm: Asynchronous Gibbs is well-suited for models where the dimensionality of the parameter space increases with sample size. In this paper we analyze Asynchronous Gibbs theoretically, and show that it performs well on both illustrative test cases and a real large-scale Bayesian application.

## 2 The Problem

In fitting a Bayesian model, calculating a posterior distribution  $\pi(\boldsymbol{\theta})$  – here and below, for notational simplicity we generally suppress conditioning on data  $\mathbf{D}$  – involves solving integrals that are often intractable. For the past two decades, the standard approach has been *Markov Chain Monte Carlo* [15, 22], which involves drawing random samples from  $\pi(\boldsymbol{\theta})$  by constructing a Markov Chain that converges to  $\pi$ . One way to create such a Markov Chain is to specify the *full conditional* distributions and construct a (sequential-scan) *Gibbs sampler* [12, 13], which consists of repeated sequential sampling of the following full conditional distributions:

$$\theta_1^* \sim \pi(\theta_1 \mid \theta_2, \dots, \theta_K) \quad \theta_2^* \sim \pi(\theta_2 \mid \theta_1^*, \theta_3, \dots, \theta_K) \quad \dots \quad \theta_K^* \sim \pi(\theta_K \mid \theta_1^*, \dots, \theta_{K-1}^*). \quad (1)$$

It has been shown [13] that under weak regularity conditions this Markov chain has  $\pi(\boldsymbol{\theta})$  as its stationary distribution. It is also possible – and sometimes preferable [28] – to draw the samples from the full conditional distributions in random order, leading to a *random-scan Gibbs sampler* [21].

A key property is that of *synchronicity*: each full-conditional sample is completed before the next begins. This causes the algorithm to parallelize poorly, for the following reasons.

- (a) *Inherently sequential nature*: to draw the next full conditional, the algorithm must know the values of *all of the most recent* full conditionals – it proceeds one step at a time.
- (b) *Centralization*: all of the data typically needs to be present on one computer.

Here we present an extension of Gibbs sampling that attacks these problems by running an equivalent computation fully in parallel on a cluster.

## 3 Previous work on Bayesian computation at scale

There are a number of approaches for Bayesian computation in parallel and distributed environments. Much recent work takes a *divide-and-conquer* approach. With these methods, the (large) data set is broken into small pieces (*shards*), with MCMC performed on each shard on a different processor in parallel and with the results averaged together to produce what is hoped is a good approximation to the (possibly intractable) MCMC calculations on the entire data set. Variations have been considered by Scott et al. [32], Neiswanger et al. [26], Srivastava et al. [33], and others. These methods solve

the synchronization problem by *not communicating at all* – our approach is fundamentally different, as it involves *communicating as much as possible*.

Asynchronous Gibbs has been analyzed under specific settings by various authors. Ihler and Newman [17] proposed a special case of our algorithm, applied to the topic of latent Dirichlet allocation (LDA). They derived a bound on the difference between the Monte Carlo output proposed by their sampling scheme and the output of an LDA-specific standard Gibbs sampler. Johnson et al. [19] have analyzed our approximate algorithm restricted to Gaussian targets, proved that – if the Gaussian target’s precision matrix is diagonally dominant – approximate Asynchronous Gibbs converges to the correct mean, and demonstrated some connections with parallel algorithms for solving linear systems. De Sa et al. [7] has analyzed the asymptotic bias of our approximate algorithm under much stronger regularity conditions.

## 4 Asynchronous Gibbs Sampling

Asynchronous Gibbs sampling is an algorithm for performing MCMC inference in parallel, without synchronization or locking. To understand the algorithm, we introduce the following concepts and notation:

- $w_i$ : a worker that performs computations, such as a computer on a network.
- $\theta$ : the parameter vector from whose posterior distribution we wish to sample.
- $\theta_i$ : a subset of  $\theta$  that is assigned to worker  $w_i$ .
- $\theta_{ci} \mid \theta_{-ci}$ : a full conditional random variable or block of random variables contained in  $\theta_i$ . Full conditional in this context means the most recent values of all other variables in  $\theta$  that are *known to worker  $w_i$*  (which may not be the most recent values *known to the cluster as a whole*, due to network delays, packet loss, outages, and other scenarios).

Asynchronous Gibbs sampling then proceeds as follows.

- (a) Provide all workers with initial values of the chain.
- (b) For each worker, repeat the following in parallel without synchronization:
  - (i) Select a variable or block  $\theta_{ci}$  from  $\theta_i$  at random with constant probability  $0 < p_{\theta_{ci}} < 1$ , exactly as in random-scan Gibbs.
  - (ii) Sample  $\theta_{ci} \mid \theta_{-ci}$ .
  - (iii) Send the update  $\theta_{ci}$  to all the other workers.
  - (iv) Process all updates received from other workers and proceed to the next iteration.

This is similar to standard random-scan Gibbs sampling, but with one primary difference: instead of sampling all the variables conditional on the *most recent values*, each worker is sampling conditional on the *most recent values that it knows about*. Note that in full generality, there are no restrictions on the way in which variables are sampled: Metropolis [22] and Metropolis-adjusted Langevin [29] steps are allowed, as are Slice Sampling [25] steps, and virtually all other techniques, as long as they can be used together to form a valid MCMC algorithm (see Section 5.1). Also note that not all variables need to be transmitted – it is only necessary that each worker is *able* to either sample

or receive each variable. In practice, we have found it effective to partition and transmit latent variables that correspond to data points, and sample – locally on every worker – variables located at the top level of a hierarchical model. We focus here on the case where all transmitted variables are sampled via Gibbs steps.

There are two ways in which updates received from other workers may be processed:

- Exact: accept updates with probability  $\min \left\{ 1, \frac{f(\boldsymbol{\theta}_{\text{new}})f(\boldsymbol{\theta}_{\text{old}} \mid \boldsymbol{\theta}_{\text{sender}})}{f(\boldsymbol{\theta}_{\text{old}})f(\boldsymbol{\theta}_{\text{new}} \mid \boldsymbol{\theta}_{\text{sender}})} \right\}$  – see Section 5.1.
- Approximate: accept all updates.

The remarkable property we have observed is that, in many situations, the Metropolis-Hastings (MH) acceptance probability is close to 1 sufficiently often that the approximate algorithm yields the same numerical answer as the exact algorithm up to Monte Carlo noise. This allows us to ignore the MH correction while still obtaining good numerical results: intuitively, we’re replacing the MH acceptance ratio with a biased estimator of it, namely 1. This type of approximation can sometimes be justified from a *Noisy Monte Carlo* perspective – see Alquier et al. [1].

In Section 5.1 we prove convergence for the exact algorithm. In Sections 6.1 - 6.2 we illustrate the approximate algorithm, which has much better scaling properties. In Section 6.3, we examine a case where the approximate algorithm can fail, and discuss when using it is appropriate.

If exactness is desired, the MH ratio can be computed quickly and inexpensively in a wide variety of problems – in Section 5.2, we show how it can be calculated in exchangeable latent variable models using only one data point. Unfortunately, this can involve transmitting data – thus, we recommend running the approximate algorithm and collecting a random sample of the MH acceptance probabilities while the algorithm unfolds. This sample can be examined to see whether its distribution is sufficiently concentrated around 1. This gives a diagnostic check for whether the approximate algorithm is appropriate – see Section 6.3.

Our algorithm possesses a variety of positive characteristics with respect to parallelism, especially in a cluster setting. It does not require any synchronization of  $\boldsymbol{\theta}$ . Furthermore, it is fault-tolerant – if a variable is dropped, due for instance to network traffic congestion, the workers simply continue, without affecting asymptotic convergence.

In many practical situations, we find that the algorithm scales superlinearly with respect to workers. Intuitively, if we have  $m$  workers and  $p$  full conditional distributions, it will take each worker approximately  $\frac{p}{m}$  iterations to either sample or receive each full conditional on average once – this is much smaller than the  $p$  iterations required under independent parallel chains. However, parallelism can both accelerate and slow down mixing at the same time in different ways – see De Sa et al. [7] – so determining how the algorithm scales is difficult. Nonetheless, in many situations, we have observed that if we double the number of workers, Asynchronous Gibbs will be more than twice as fast.

Significant care must be taken to properly tune the algorithm and monitor convergence. The standard “effective sample size” calculation used for MCMC diagnostics does not immediately extend to multiple workers, because their chains are not independent. We have found it helpful in practice to examine the sample autocorrelation function (ACF) for each worker at large time lags, on the order of thousands or tens of thousands of observations: the point is to check whether the ACF goes to 0 at a number of lags that is lower than the total number of Monte Carlo iterations performed on its corresponding worker.

Our algorithm is implemented in *Scala*, a compiled language similar to and interoperable with *Java* that is well equipped for parallel processing use cases. Network communication and cluster management are handled by *Akka*, a decentralized actor-based message-passing framework written in *Scala* for large-scale distributed applications. Numerical computation is done via *Breeze*, a *Scala* library written for fast and accurate computation, designed for natural language processing and scientific computing. With slight modifications, our implementation could run on large-scale distributed data sets in the *Hadoop* environment. However, since Asynchronous Gibbs sampling is not a deterministic algorithm, it does not translate directly into the *MapReduce* paradigm.

## 5 Convergence and Properties of the Algorithm

**5.1. Convergence.** In this section we prove that, provided we start with a well-defined Markov Chain, Asynchronous Gibbs sampling will converge to the correct target distribution. Our strategy is two-fold. Firstly, we define a synchronous parallel MCMC algorithm that formalizes the way in which workers draw samples and communicate with one another, under the assumption that communication is instantaneous and synchronous, using ideas inspired by the coupling of chains in *parallel tempering* [35]. Then we note that MCMC methods belong to the class of *fixed-point* algorithms, and hence we can use a result from the asynchronous convergence of these algorithms, due to Baudet [3] and Bertsekas [4], to prove that the asynchronous version of our parallel algorithm with non-instantaneous communication converges as well.

Our notation is as follows:

- $w_i$ : an arbitrary worker.
- $k$ : the current iteration of the chain.
- $p$ : the total number of full conditional distributions.
- $m$ : the total number of workers.
- $(\Omega, \mathcal{F}, \mathbb{P})$ : the probability triple upon which the target distribution is defined.
- $\mathcal{M}_i$ : all probability measures on  $\Omega$ . Note that (trivially)  $\mathcal{M}_i = \mathcal{M}_{i'}$  for all  $i, i'$ .
- $\mathcal{M} \triangleq \times_{i=1}^m \mathcal{M}_i$ : the product of probability measures on  $\Omega^m$ .
- $\pi$ : the measure corresponding to the target density, i.e., the posterior distribution  $\pi(\boldsymbol{\theta} \mid \mathbf{D})$ .
- $\Pi \triangleq \times_{i=1}^m \pi$ : the product of  $m$  measures  $\pi$ .
- $\theta_{ci}$ : the full conditional random variable corresponding to coordinate  $c$  on worker  $i$ .
- $\|\cdot\|_{\text{TV}}$ : the total variation metric defined on  $\mathcal{M}_i$ .
- $\|\cdot\|_{\text{TV}}^\times \triangleq \sum_{i=1}^m \|\cdot\|_{\text{TV}}$ : a metric on  $\mathcal{M}$  defined via the sum of  $m$  total variation metrics.

We begin by defining the MCMC algorithm that we wish to parallelize.

**Definition 1 (Underlying Chain).**

Let the underlying chain be a well-defined first-order Markov operator  $P$  satisfying the following:

- Stationarity:  $P(\pi) = \pi$ .

- Convergence:  $\|P^k(\cdot) - \pi\|_{\text{TV}} \rightarrow 0$  as  $k \rightarrow \infty$ .
- Gibbs Kernel:  $P$  is constructed via full conditional distributions  $f(\theta_c \mid \theta_{-c})$ .

Conditions for stationarity and convergence typically follow from  $\phi$ -irreducibility, aperiodicity, and related assumptions required by standard MCMC schemes.

We now define a Markov chain on an extended space that captures the behavior of our algorithm *if communication is instantaneous*, i.e., if there are no asynchronous delays.

**Definition 2 (Synchronous Parallel Chain).**

Let the synchronous parallel chain be a first-order Markov operator  $H : \mathcal{M} \rightarrow \mathcal{M}$  with components  $H_i : \mathcal{M}_i \rightarrow \mathcal{M}_i$  for each worker  $w_i$ . Define  $H$  according to a Metropolis-Hastings transition as follows:

- (i) Randomly select a worker  $w_s$ .
- (ii) Randomly select a full conditional coordinate  $c$  from the set of coordinates that  $w_s$  works on.
- (iii) Propose  $\theta'_{cs}$  from  $f(\theta_{cs} \mid \theta_{-cs})$ .
- (iv) For each worker  $w_i$ , accept the proposed move with probability

$$\alpha_i = \min \left\{ 1, \frac{f(\theta'_{cs}, \theta_{-ci}) f(\theta_{ci} \mid \theta_{-cs})}{f(\theta_{ci}, \theta_{-ci}) f(\theta'_{cs} \mid \theta_{-cs})} \right\}, \quad (2)$$

staying at the previous value otherwise.

Thus the synchronous parallel chain selects a worker at random and proposes from that worker's full conditional *at every worker*. Note that  $\alpha_s = 1$ , because on worker  $w_s$  – whose full conditional was selected – the proposal is exactly a Gibbs step and is hence always accepted. We now show that if each worker performs the MH test independently given the proposal, the resulting chains also converge. To do so, we begin with a lemma.

**Lemma 3 (Convergence of Metropolis-Coupled Chains).**

Suppose that  $Q$  is a Markov Chain with stationary and unique limiting distribution  $\pi$ . Suppose that  $R$  is a Markov Chain with a Metropolis-Hastings proposal centered at the current value of  $Q$ . Then  $R$  has stationary and unique limiting distribution  $\pi$ .

*Proof.* Let  $R_\eta$  be a Metropolis-Hastings independence sampler with proposal centered at  $\eta$  and acceptance probability chosen to ensure stationarity of the chain at  $\pi$ , and let  $\{R_\eta\}$  be the set of all such Markov transition kernels. Then  $R$  is an adaptive MCMC algorithm in which the transition kernel  $R_\eta$  is selected based on the current value of  $Q$ . As the transition kernel of  $Q$  does not depend on the current value of  $R$ , it can be seen that  $R$  is an *independent adaptation*. It then follows from Proposition 1 in Roberts and Rosenthal [30] that  $\pi$  is a stationary distribution of  $R$ . Irreducibility, aperiodicity, and Harris recurrence of  $R$  follow immediately, as these properties must by assumption hold for  $Q$ . Thus  $R$  admits  $\pi$  as its stationary and unique limiting distribution. ■

**Proposition 4 (Synchronous Parallel Convergence).**

Each  $H_i$  admits  $\pi$  as its stationary distribution and unique limiting distribution. Furthermore,

$$\|H - \Pi\|_{\text{TV}}^\infty \rightarrow 0. \quad (3)$$

*Proof.* The proposal for the chain can be written hierarchically as follows:

$$w_s : f(\theta'_{cs} \mid \theta_{-cs}) \quad w_j : \delta(\theta'_{cs}) \text{ for } j \neq s. \quad (4)$$

In other words, on worker  $w_s$  the proposal is the selected full conditional, and on all other workers it is a point mass centered at the proposed value on  $w_s$ . Therefore, marginally on each worker, the proposal is just

$$f(\theta'_{cs} \mid \theta_{-cs}). \quad (5)$$

Then  $H_s$  is just a normal Gibbs sampler and therefore converges, and Lemma 3 implies convergence of each  $H_i$  with  $i \neq s$ . It can also be seen that

$$\|H - \Pi\|_{\text{TV}}^\times = \sum_{i=1}^m \|H_i - \pi\|_{\text{TV}} \rightarrow 0, \quad (6)$$

since convergence implies that each component of the sum goes to zero. ■

Note that our definitions are specifically created to track convergence marginally on each worker, and no attempt is made to analyze dependence between different workers.

We now move to the second stage of the proof. From here, we would like to show that  $H$  converges asynchronously, i.e., convergence is still valid in the setting in which each worker does not necessarily know the precise current state of all other workers, and instead works with the latest state that it knows about. We begin by stating the Frommer and Szyld [10] model of distributed computation.

**Definition 5 (Asynchronous Computation).**

Start with the following fixed-point computation problem:

- (P1) Let  $E \triangleq \times_{i=1}^m E_i$  be a product space. Index  $i$  refers to the component of  $E$  belonging to worker  $w_i$ .
- (P2) Let  $H: E \rightarrow E$  be a function (here, a Markov operator), and denote its individual components by  $H_i$ .
- (P3) Let  $x^*$  be a fixed point of  $H$ , i.e.,  $x^* = H(x^*)$ .

Now, define the following cluster computation model:

- Let  $k \in \mathbb{N}_0$  be the total number of iterations performed by all workers.
- Let  $s_i(k) \in \mathbb{N}_0$  be the total number of iterations on component  $i$  by all workers.
- Let  $I_k$  be an index set containing the components updated at iteration  $k$ .

Next, assume the following basic regularity conditions on the cluster:

- (R1)  $s_i(k) \leq k - 1$ , i.e., a worker's current state cannot be based on future values.
- (R2)  $\lim_{k \rightarrow \infty} s_i(k) = \infty$ , i.e., workers don't stop permanently mid-run.
- (R3)  $|\{k \in \mathbb{N} : i \in I_k\}| = \infty$ , i.e., no component stops being updated or communicated.

Finally, define  $x^k$  component-wise via the following:

$$x_i^k \triangleq \begin{cases} H_i(x_1^{s_1(k)}, \dots, x_m^{s_m(k)}) & \text{for } i \in I_k, \\ x_i^{k-1} & \text{otherwise.} \end{cases} \quad (7)$$

Then  $x^k$  is termed an *asynchronous iteration*, and  $\{x^k : k \in \mathbb{N}_0\}$  is termed an *asynchronous computation*.

The above definition is broad enough to encompass almost every asynchronous distributed computation that possesses any hope of convergence, and its requirements should be trivially true in any such computation. In particular, it does not make sense to even think about convergence in situations where work on some portion of the problem stops prematurely and permanently, violating (R2), or in situations where there does not exist a way to split up the problem among the workers.

With this computational model in mind, the following general theorem gives a sufficient set of conditions under which the asynchronous iterates  $x^k$  converge to the correct answer.

**Result 6 (Asynchronous Convergence).**

Given a well-defined asynchronous computation as in Definition 5, assume the following conditions hold for all  $k \in \mathbb{N}_0$ :

- (C1) There are sets  $E^k \subseteq E$  satisfying  $E^k = \bigtimes_{i=1}^m E_i^k$  (*box condition*).
- (C2) For  $E^k$  in (C1),  $H(E^k) \subseteq E^{k+1} \subseteq E^k$  (*nested sets condition*).
- (C3) There exists an  $x^*$  such that  $y^k \in E^k \implies y^k \rightarrow x^*$  in some metric (*synchronous convergence condition*).

Then  $x^k \rightarrow x^*$  in the same metric.

*Proof.* See Frommer and Szyld [10], Bertsekas [4], and Baudet [3]. ■

This result can be used to prove that a wide variety of asynchronous iterative algorithms converge. In particular, we believe that it can be used to construct an alternative proof that the *Hogwild* scheme (asynchronous stochastic gradient descent) [27] converges, as well as for asynchronously parallelizing other iterative stochastic algorithms such as EM [8].

To write a proof using the above result, the main thing that needs to be shown is that the state space for the iterative algorithm in question can be partitioned into a sequence of boxes that shrink toward the fixed point in question in a sequentially continuous way as the iterative operation is applied. Intuitively, once such a sequence is defined, the *Banach Fixed Point Theorem* guarantees that it will eventually land within a neighborhood of the fixed point. Within the context of Asynchronous Gibbs sampling, this amounts to proving that a well-defined standard Gibbs sampler can be partitioned according to (P1) and passes conditions (C1–C3) described above. We need the following basic regularity assumption.

**Assumption 7 (No Worker Dies).**

All of the following hold:

- (i) At least one worker is assigned to each full conditional  $\theta_c \mid \theta_{-c}$  defined in the underlying chain.
- (ii) For each worker  $w_i$  and each coordinate  $c$ , the sequence  $\theta_{ci} \mid \theta_{-ci}$  is infinite.
- (iii) For each worker  $w_i$ , coordinate  $c$ , and iteration  $s_i(k)$ ,  $\theta_{ci}^k$  is updated conditional on  $\theta_{ci}^{k'}$ , where  $k' \leq k - 1$ .



*Justification.* Condition (i) ensures that, taken together, the workers actually work on the full problem. Condition (ii) is needed because without it there will be some dimension that stops either being updated or communicated after finite time. Note that in practice, if a worker does crash, it suffices to restart the worker sufficiently quickly. Condition (iii) ensures that workers cannot receive updates from the future.

We need the following result from Markov Chain theory.

**Result 8 (Monotonic Convergence).**

Suppose that  $P$  is a Markov operator with unique stationary distribution  $\pi$  and initial distribution  $\mu \in \mathcal{M}$ . Then

$$\|P^{k+1}(\mu) - \pi\|_{\text{TV}} \leq \|P^k(\mu) - \pi\|_{\text{TV}}. \quad (8)$$

*Proof.* Meyn and Tweedie [23], Proposition 13.3.2. It can be seen that this also holds for the metric  $\|\cdot\|_{\text{TV}}^\times$  by applying its definition. ■

We now proceed with the proof.

**Lemma 9 (Box Condition).**

Take  $E = \mathcal{M}$  and  $E_i = \mathcal{M}_i$ . Fix the initial distribution  $\mu \in \mathcal{M}$ . Define the following:

$$E^k \triangleq \{\nu \in \mathcal{M} : \|\nu - \Pi\|_{\text{TV}}^\times \leq \|H^k(\mu) - \Pi\|_{\text{TV}}^\times\}. \quad (9)$$

Then there exist sets  $E_i^k$  such that  $E^k = \times_{i=1}^m E_i^k$ .

*Proof.* From the definitions we know that  $E = \times_{i=1}^m E_i$ . By construction,  $E^k \subseteq E$ . Let  $E_i^k = E^k \cap E_i$ . Because all of these sets are by definition nonempty, and the sets  $E_i$  form a partition of  $E$  (since  $\mathcal{M} = \times_{i=1}^m \mathcal{M}_i$ ), we get  $E^k = \times_{i=1}^m E_i^k$ . ■

**Lemma 10 (Nested Sets Condition).**

Let  $E^k$  be defined as in the previous lemma. Then  $H(E^k) \subseteq E^{k+1} \subseteq E^k$ .

*Proof.* Fix  $\nu$  and consider  $E^k$ . By monotone convergence,  $\|H^k(\mu) - \Pi\|_{\text{TV}}^\times$  is non-increasing in  $k$ , and we get  $E^{k+1} \subseteq E^k$ . We also have  $E^{k+1} = H(E^k)$  by construction. ■

**Theorem 11 (Asynchronous Gibbs Converges).**

Assume all assumptions and theorems above. Then Asynchronous Gibbs sampling converges to  $\pi$  on each worker in total variation.

*Proof.* Below, we verify that all of the conditions required in Result 6 (Asynchronous Convergence) hold.

(P1) Take  $E = \mathcal{M}$ .

(P2) Take  $H$  as defined in the Synchronous Parallel Chain Definition.

(P3) Take  $x^* = \Pi$ .

(R\*) All satisfied by the No Worker Dies Assumption.

(C1) Satisfied by the Box Condition Lemma.

(C2) Satisfied by the Nested Sets Condition Lemma.

(C3) Satisfied by the Synchronous Parallel Convergence Theorem.

Now invoke the Asynchronous Convergence Result to conclude that the distribution from the combined Asynchronous Gibbs sampling chain across all workers converges to  $\Pi$  in total variation in each component, and thus each worker's chain converges to  $\pi$  marginally. ■

We note in conclusion that this proof does not require any structure on the underlying chain, beyond the basic assumptions required for it to converge. In the example in Section 6.1, for instance, each worker updates all of its assigned variables  $\theta_i$  in a single block.

**5.2. Exchangeable Latent Variable Models and Exact Asynchronous Gibbs.** If we are interested in sampling from an exchangeable latent variable hierarchical Bayesian model, the posterior ratio used in the MH acceptance test in exact Asynchronous Gibbs simplifies to an expression involving only one data point – this means that this ratio can be evaluated locally to each worker in a parallel environment. To illustrate, consider the following model:

$$y_i \mid \nu_i \propto A(\nu_i) \quad \nu_i \mid \theta \stackrel{\text{iid}}{\sim} B(\theta) \quad \theta \propto \pi(\theta), \quad (10)$$

in which  $A$  and  $B$  are arbitrary distributions and (in this section only)  $\pi$  no longer refers to the posterior distribution of interest. We can define a Gibbs sampler of the form

$$\nu_i \mid \theta, y_i \sim C(\theta, y_i) \quad \theta \mid \nu_1, \dots, \nu_N \sim D(\nu_1, \dots, \nu_N), \quad (11)$$

where  $C$  and  $D$  are arbitrary distributions. Assume that we can sample from  $C$  directly. This model's posterior distribution has the form

$$\left[ \prod_{i=1}^N f(y_i \mid \nu_i) f(\nu_i \mid \theta) \right] \pi(\theta) \propto \prod_{i=1}^N f(\nu_i \mid y_i) f(\theta \mid \nu_i) \propto \prod_{i=1}^N f(\theta, \nu_i \mid y_i). \quad (12)$$

Now define an Asynchronous Gibbs sampler in which all workers transmit the values of their corresponding  $\nu_i$  but never transmit  $\theta$ . Consider a transmitted update from  $\nu_j$  to  $\nu'_j$ . Let  $q$  be the full conditional proposal distribution on the worker that sent  $\nu'_j$ , and assume that this worker transmits the parameters of that distribution along with  $\nu'_j$ . Notice that since  $q$  is a full conditional distribution, it does not depend on  $\nu_j$  or the previous value of  $\nu'_j$  on the transmitting worker. Then the MH acceptance probability takes the form:

$$\min \left\{ 1, \frac{f(\theta, \nu'_j \mid y_j) \prod_{i \neq j} f(\theta, \nu_i \mid y_i) q(\nu_j)}{f(\theta, \nu_j \mid y_j) \prod_{i \neq j} f(\theta, \nu_i \mid y_i) q(\nu'_j)} \right\} = \min \left\{ 1, \frac{f(\theta, \nu'_j \mid y_j) q(\nu_j)}{f(\theta, \nu_j \mid y_j) q(\nu'_j)} \right\}. \quad (13)$$

Thus we can carry out the evaluation using only one data point. The details for doing so are problem-specific and depend on how the data is stored. For example, if  $y_j$  is not available on other workers, we can transmit it over network along with  $\nu'_j$ . If  $\nu_j$  is also not available on other workers, but the latent variables  $\nu_j$  form a non-overlapping partition among the workers, then we can transmit  $(\nu'_j, \nu_j, y_j, q)$ , because  $\nu_j$  can only be updated on other workers through communication. This situation occurs in some problems where parameters – such as  $\theta$  in Equation (10) – that are located at the top of a hierarchical model may depend on  $\nu_j$  only through sufficient statistics, and

where storing  $\nu_j$  for all  $j$  on every worker is thus unnecessary. These details illustrate the flexibility that Asynchronous Gibbs sampling gives the user in handling large distributed data sets.

Note that if the data points  $y_j$  are sufficiently large, transmitting them may be too expensive. We instead recommend computing and storing the MH ratios at random with small probability, and using them as a convergence diagnostic – see Section 6.3.

## 6 Examples

**6.1. Example: Gaussian Process regression, a highly simplified spatial model with  $n = 71,500$ .** This problem originally appeared in an example due to Neal [24] – we examine a variation due to Kottas [20]. In this example, we used our algorithm to sample from the posterior distribution arising from a simple Gaussian Process regression problem. Our method is close to exact – inexactness is introduced only through a block matrix inverse approximation scheme, and through approximate Asynchronous Gibbs.

This example is far too simple for use in a real spatial statistics problem – rather, we present it as a way to illustrate how approximate Asynchronous Gibbs sampling can be used for computation at scale. Our goal was to reconstruct the following function:

$$\tilde{f}(x) = 0.3 + 0.4x + 0.4 \sin(2.7x) + \frac{1.1}{1+x^2} \quad x \in [-3, 3]. \quad (14)$$

This is then reflected and copied around the lines  $x = 3, 9, \dots$ , and  $x = -3, -9, \dots$ , in such a way that  $\tilde{f}(x)$  becomes periodic with period 6 and is continuous everywhere. To simplify our example, we assumed that our data lives on a grid with spacing equal to 0.06 (i.e.,  $x_1 = 0, x_2 = 0.06, x_3 = -0.06, \dots$ ). To generate the data, we added Gaussian white noise with standard deviation 0.2. Our model for reconstructing this function is then defined in the following way:

$$y_i = f(x_i) + \varepsilon_i \quad f(x_i) \sim \text{GP} \quad \varepsilon_i \stackrel{\text{iid}}{\sim} \text{N}(0, \sigma^2). \quad (15)$$

Here  $i = 1, \dots, n = 71,500$  with  $x$  on  $[-2,145, 2,145)$ . For simplicity, we selected a Gaussian Process with constant mean function  $\mu$  and exponential covariance function  $-\tau^2 \exp\{-\phi|x - x'|\}$ , together with the following hyperpriors:

$$\mu \sim \text{N}(a_\mu, b_\mu) \quad \sigma^2 \sim \text{IG}(a_\sigma, b_\sigma) \quad \tau^2 \sim \text{IG}(a_\tau, b_\tau) \quad \phi \sim \text{U}(0, b_\phi). \quad (16)$$

By introducing latent variables  $\theta_i$  corresponding to each data point, the model can then be expressed in the following way:

$$y_i \mid \theta_i, \sigma^2 \sim \text{N}(\theta_i, \sigma^2) \quad \boldsymbol{\theta} \sim \text{N}_n(\mu \mathbf{1}_n, \tau^2 \mathbf{H}(\phi)) \quad H_{ij}(\phi) = \exp\{-\phi|x_i - x_j|\}. \quad (17)$$

By conjugate updating, this yields Inverse-Gamma posteriors for  $\sigma^2$  and  $\tau^2$ , a normal posterior for  $\mu$ , and an  $n$ -dimensional multivariate normal posterior for  $\boldsymbol{\theta}$ . Since  $\phi$  is non-conjugate, to simplify things for our example we fixed it at 0.5, which is an interpretable value that is close to its MLE. If  $n$  is large, block sampling from this posterior is intractable because it requires the frequent inversion of two  $(n \times n)$  matrices. It is possible to integrate  $\boldsymbol{\theta}$  out of the model, but this does not avoid large matrix inversion. We propose the following scheme to sample from the posterior of  $(\mu, \sigma^2, \tau^2, \boldsymbol{\theta})$ .

In our approach, we update individual slices of  $\boldsymbol{\theta}$ , consisting of 500 elements, via Gibbs steps. To do this, we sample from full conditional distributions of the form  $\boldsymbol{\theta}_{1:500} \mid \boldsymbol{\theta}_{501:n}, \mu, \sigma^2, \tau^2$  for arbitrary indices (recall that  $\phi$  is fixed). Thus we need to sample from conditional Gaussian distributions of portions of  $\boldsymbol{\theta}$ , given the rest of  $\boldsymbol{\theta}$ . To do this without ever constructing the large covariance matrix, which may be too big to store in memory, we need to be able to invert  $\mathbf{H}(\phi)$ , multiply by  $\tau^{-2}$ , add  $\sigma^{-2} \mathbf{I}_n$ , and invert back. The following scheme allows us to do this elementwise, with only one approximate inversion along the way, which can with further work likely be refined into an exact inversion.

Since we have made the simplifying assumption that our grid is evenly spaced, the covariance matrix  $\mathbf{H}(\phi)$  is Toeplitz. Additionally, since our covariance function is exponential, the resulting covariance matrix is hyperbolic, and can be inverted element-wise analytically via a technique due to Dow [9], with inverse that simplifies to

$$\mathbf{H}^{-1}(\phi) = \begin{bmatrix} d_0 & a & 0 & \dots & \dots & \dots & 0 \\ a & b & a & 0 & \ddots & \ddots & \vdots \\ 0 & a & b & \ddots & \ddots & \ddots & \vdots \\ \vdots & 0 & \ddots & \ddots & \ddots & 0 & \ddots \\ \vdots & \ddots & \ddots & \ddots & b & a & 0 \\ \vdots & \ddots & \ddots & 0 & a & b & a \\ 0 & \dots & \dots & \dots & 0 & a & d_0 \end{bmatrix} \quad \begin{cases} b = -\coth(-\phi\rho) \\ a = \frac{\text{csch}(-\phi\rho)}{2} \\ d_0 = \frac{e^{-\phi\rho(2N-3)} \text{csch}(-\phi\rho) + 1 - \coth(-\phi\rho)}{2 - 2e^{-\phi\rho(2N-3)}} \\ \rho = \text{grid spacing size} = 0.06 \\ N = \text{dimension of } \mathbf{H}(\phi) \end{cases} \quad (18)$$

Note that this  $\mathbf{H}^{-1}(\phi)$  is tridiagonal with modified corner elements. While this technique limits the generality of our Gaussian Process prior, more complicated ways of avoiding large matrix inversions are available with modern spatial priors such as Nearest Neighbor Gaussian Processes [2].

Next, we multiply by  $\tau^{-2}$ , and add  $\sigma^{-2}$  to the diagonal. The resulting covariance matrix is still tridiagonal with modified corner elements. We do not know how to invert this matrix analytically, but we do know how to invert the general tridiagonal Toeplitz matrix without modified corner elements, via a technique due to Hu and O’Connell [16]. We approximate the tridiagonal form by assuming that  $d_0 \doteq b$  in (18) – this works well except at the points where the partition slices of  $\boldsymbol{\theta}$  join, where a small amount of error is introduced.

Finally, to find the mean vector, we need to multiply the covariance matrix defined by (18) by a term that includes the full data. Since the data set can be too large to store on a single machine, this is intractable. However, not all is lost: since the resulting covariance function decreases rapidly to 0 as the distance away from the full conditional of interest increases, this multiplication can be carried out to arbitrary precision by simply taking a slice in the center of the matrix in a neighborhood around the full conditional of interest, avoiding use of the full data. This idea also underlies *covariance tapering* [11] and *composite likelihood methods* for spatial problems [34].

After all of these steps, we can sample any slice of  $\boldsymbol{\theta}$  full conditionally via the standard Schur complement formula, since the full conditional of a Gaussian is Gaussian.

With standard Gibbs, this technique is still intractable: there are too many full conditionals to sample for the chain to produce its output in a reasonable amount of time. Asynchronous Gibbs lets us parallelize this computation, producing results in minutes (given sufficient hardware) that would have taken days or weeks with standard MCMC. In this example we used 143 workers

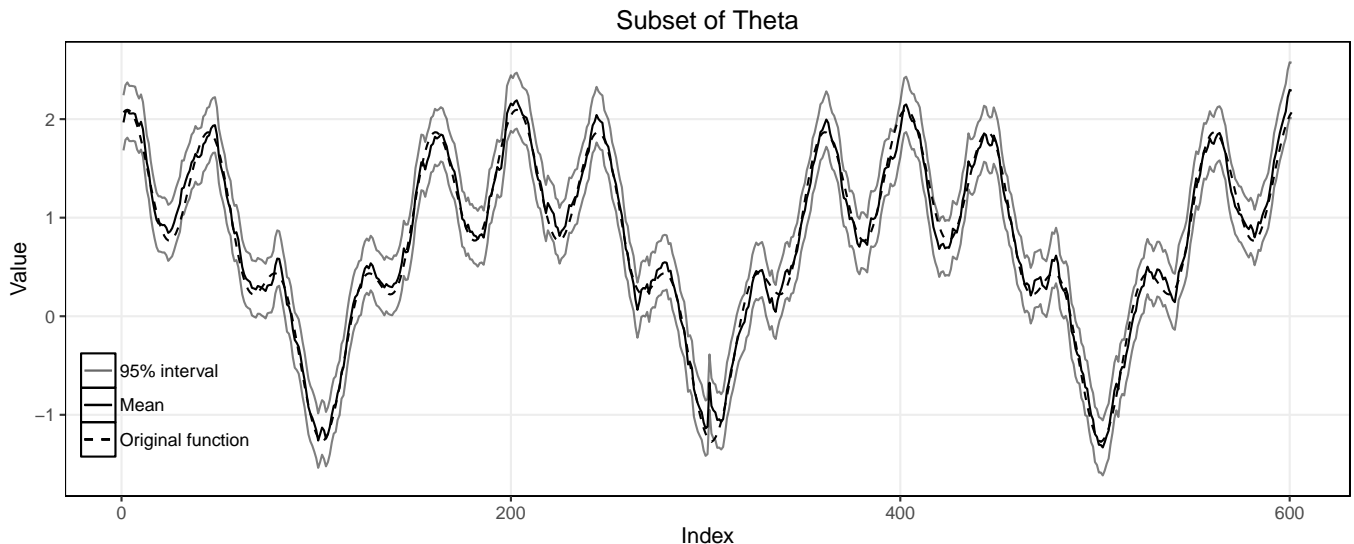


Figure 1: Partial subset of  $\theta$  for two workers (split at center), in the Gaussian Process regression example of Section 6.1.

with 1 CPU each. Each worker was responsible for 500 values of  $\theta$  (different from those handled by the other workers), and for  $(\mu, \sigma^2, \tau^2)$ . We started our algorithm from garbage initial values:  $\mu = 10, \sigma^2 = 10, \tau^2 = 10, \theta = \mathbf{0}$ . Our algorithm converged rapidly to within a small neighborhood of the correct solution. It finished, producing approximately 10,000 samples per worker, in around 20 minutes.

In Figure 1 we plot a slice of the data, together with the correct solution. As noted above, our matrix inversion approximation scheme is inaccurate around the edges of each slice of  $\theta$  (this can be seen in the middle of Figure 1, at index 300), and hence these values are not as accurate as those elsewhere. The algorithm converged in an analogous fashion for all other slices of the data.

All of the components of  $(\mu, \sigma^2, \tau^2)$  are relatively easy to learn. We have observed similar results for a wide variety of statistical models in the large-data setting: parameters that depend on the full data tend to be easy to learn, with low posterior variance, and can often safely be fixed at their MLEs if doing so helps computation. Note that if we had not fixed  $\phi$ , we would have needed to compute a large matrix expression involving  $\mathbf{H}^{-1}(\phi)$  in its entirety for every sample of  $\tau^2$  and  $\mu$ . Because  $\mathbf{H}^{-1}(\phi)$  is available analytically, this is tractable, but for simplicity, we chose to fix  $\phi$  instead. This choice is reasonable given that  $\phi$  and  $\tau^2$  are jointly weakly identifiable.

We conclude that Asynchronous Gibbs sampling is a promising algorithm for Gaussian Process regression models at scale.

**6.2. Example: Mixed-effects regression, a complex hierarchical model with  $n = 1,000,000$ .** The following model, due to von Brzeski et al. [37], was used in a large-scale analysis of product updates at eBay Inc. Because users choose when to update to the latest version of the product, analysis of product updates is done not by controlled experiment but by observational study, and causal inference is difficult. In particular, it is necessary to control for the *early-adopter effect*, in which the behavior of the response is correlated with how quickly a user adopts the treatment after release. To adjust for this effect, a Bayesian hierarchical mixed-effects regression model was selected. Since we are primarily interested in the computational aspects of this problem, we omit further discussion of how

and why the particular model was selected and interpretation of the results – such discussion can be found in the original publication.

A variety of different data sets have been used with this model – the data set that we employed, selected for convenience, consists of  $n = 1,000,000$  users. The model can be written in the following way:

$$\mathbf{y}_i = \mathbf{F}_i \boldsymbol{\beta}_i + \mathbf{W}_i \boldsymbol{\gamma} + \boldsymbol{\varepsilon}_i \quad \boldsymbol{\beta}_i \mid \boldsymbol{\mu}, \boldsymbol{\Sigma} \stackrel{\text{iid}}{\sim} \text{N}_d(\boldsymbol{\mu}, \boldsymbol{\Sigma}) \quad \boldsymbol{\varepsilon}_i \mid \nu \stackrel{\text{iid}}{\sim} \text{N}_{T-p}(\mathbf{0}, \nu \mathbf{I}_{T-p}), \quad (19)$$

It uses the data  $\mathbf{y}_i : (T - p) \times 1$ ,  $\mathbf{F}_i : (T - p) \times d$ , and  $\mathbf{W}_i : (T - p) \times (T - p)$ , as well as the parameters  $\boldsymbol{\beta}_i : (d \times 1)$ ,  $\boldsymbol{\gamma} : (T - p) \times 1$ ,  $\boldsymbol{\mu} : (d \times 1)$ ,  $\boldsymbol{\Sigma} : (d \times d)$ , and  $\nu$  : scalar. It has following priors:

$$\boldsymbol{\mu} \sim \text{N}_d(\mathbf{0}, \kappa_\mu \mathbf{I}_d) \quad \boldsymbol{\Sigma} \sim \text{IW}_d(d + 1, \mathbf{I}_d) \quad \boldsymbol{\gamma} \sim \text{N}_{T-p}(\mathbf{0}, \kappa_\gamma \mathbf{I}_{T-p}) \quad \nu \sim \text{IG}(\epsilon/2, \epsilon/2). \quad (20)$$

Here  $i = 1, \dots, n$  indexes individual data points (eBay users),  $\mathbf{y}_i$  is a vector of values representing customer satisfaction for user  $i$  over time (aggregated to the weekly level),  $\mathbf{F}_i$  and  $\mathbf{W}_i$  are user-specific matrices of known constants (fixed effects),  $d$  is the length of the random-effects vector,  $T = 52$  is the number of weeks of data for each user,  $p$  is the number of lags of autoregression in the model (typically no more than 5), and  $\kappa_\mu, \kappa_\gamma, \epsilon$  are fixed hyperparameters.

The full conditionals can be sampled using a standard sequential-scan Gibbs sampler. The variables  $\boldsymbol{\beta}_{i=1, \dots, n}$  can be treated as a block and updated in parallel, since they are all conditionally independent given  $\boldsymbol{\mu}, \boldsymbol{\Sigma}, \boldsymbol{\gamma}, \nu$ .

Unfortunately, this parallelization scheme extends poorly from the multicore setting to the cluster setting, for a variety of reasons. In particular, the cluster must wait for all of the nodes to finish updating the  $\boldsymbol{\beta}_i$  before proceeding with updating other variables, and block updating  $\boldsymbol{\beta}$  does not eliminate synchronization costs. Similarly, if a single node performing any task goes offline, the entire algorithm stops – thus fault tolerance is a concern. Finally, the full conditionals for  $\boldsymbol{\mu}, \boldsymbol{\gamma}, \boldsymbol{\Sigma}$ , and  $\nu$  involve the full conditional sufficient statistics

$$\begin{aligned} \bar{\boldsymbol{\beta}} &= \frac{1}{n} \sum_{i=1}^n \boldsymbol{\beta}_i & \mathbf{S} &= \sum_{i=1}^n (\boldsymbol{\beta}_i - \boldsymbol{\mu})(\boldsymbol{\beta}_i - \boldsymbol{\mu})^T \\ \mathbf{g} &= \sum_{i=1}^n \mathbf{W}_i (\mathbf{y}_i - \mathbf{F}_i \boldsymbol{\beta}_i) & l &= \sum_{i=1}^n (\mathbf{y}_i - \mathbf{F}_i \boldsymbol{\beta}_i - \mathbf{W}_i \boldsymbol{\gamma})^T (\mathbf{y}_i - \mathbf{F}_i \boldsymbol{\beta}_i - \mathbf{W}_i \boldsymbol{\gamma}), \end{aligned} \quad (21)$$

which need to be calculated in a distributed setting – this takes significant time to compute due to network overhead.

Approximate Asynchronous Gibbs can enable this computation to be performed fully in parallel by an arbitrarily large cluster, while eliminating many of these difficulties – reducing synchronization costs and improving fault tolerance in particular.

To avoid *reduce* operations over the full data, we maintain a *cache* of  $\bar{\boldsymbol{\beta}}, \mathbf{S}, \mathbf{g}, l$ . To illustrate this, consider a new update of a single  $\boldsymbol{\beta}_i$ . When it is generated or received, the cache is updated by subtracting the portion of the sum corresponding to the old  $\boldsymbol{\beta}_i$  and adding the portion corresponding to the new value. This significantly speeds up computation, but results in higher memory use.

Each worker updates  $\boldsymbol{\mu}, \boldsymbol{\Sigma}, \boldsymbol{\gamma}, \nu$  with the same probability as each individual element  $\boldsymbol{\beta}_i$ . With 12 workers and 1,000 iterations for each  $\boldsymbol{\beta}_i$ , the algorithm generates  $12(1,000) = 12,000$  total samples for each variable. This helps with mixing, improving accuracy.

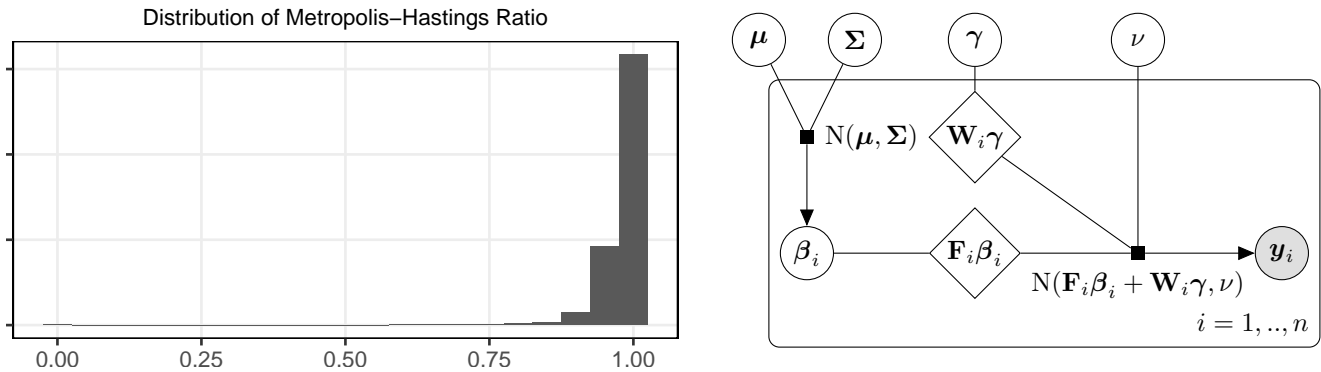


Figure 2: (Left panel) Distribution of MH acceptance probability, and (right panel) directed acyclic graph for the model, in the hierarchical mixed-effects regression example of Section 6.2.

For a fair performance comparison between approximate Asynchronous Gibbs and standard Gibbs (with multithreaded sampling of  $\beta_{i=1,\dots,n}$ ), we implemented a simple sequential-scan Gibbs sampler in *Scala*, using the exact same numerical routines (*Breeze*) as in our cluster sampler. For data size  $n = 1,000,000$  and 1,000 Monte Carlo iterations, running in parallel with 8 threads, the sequential-scan Gibbs sampler took about 12 hours to finish. Asynchronous Gibbs was much faster: with 20 workers, each with 8 threads (160 threads in total), the algorithm finished in about 1 hour.

Figure 2 gives the distribution of the MH acceptance probabilities, together with a directed acyclic graph representation of model (19–20). The probability of rejecting a random update is about 0.02, indicating that the behavior of the approximate algorithm is close to what the exact algorithm would have done (up to Monte Carlo noise) – see Section 6.3 for further discussion of this diagnostic. Both chains yielded similar diagnostic plots, which indicate issues with mixing. From a Monte Carlo accuracy standpoint, it is hard to tell whether the sequential-scan Gibbs sampler or Asynchronous Gibbs is better.

It took substantially longer for  $\nu$  to reach equilibrium with the Asynchronous Gibbs sampler: this is a result of caching. Before we implemented caching, the Asynchronous Gibbs trace plot for  $\nu$  looked similar to the sequential-scan trace plot, but the algorithm ran substantially more slowly due to time spent computing the relevant sum. Note also that caching helps to ensure that all variables take a similar amount of time to sample, which is needed to ensure that the parallel chain is approximately time-homogeneous. An example due to Murray Pollock (personal communication) has demonstrated that violating time-homogeneity in a systematic way can sometimes introduce additional bias into the algorithm.

Given the massively faster run time, Asynchronous Gibbs appears to outperform multicore sequential-scan Gibbs sampling. We were unable to acquire a cluster with more hardware on which to test our algorithm at larger scale – thus, the high end of its actual capability remains unknown. To summarize this example, Asynchronous Gibbs sampling produces output that appears to have sufficient Monte Carlo accuracy for the real-world problem at hand. The resulting chain mixes poorly, though not much more so than the standard Gibbs sampler. Thankfully, the real-world problem only demands minimal precision: the main factors of interest in the original analysis are the signs of the individual components of  $\mu$  and  $\gamma$ . The output of Asynchronous Gibbs sampling was sufficient for these purposes, and in this problem the benefits of parallelism and speed outweighed the potential costs associated with slower mixing.

**6.3. Jacobi Sampling and Approximate Asynchronous Gibbs.** We now illustrate a way in which Asynchronous Gibbs sampling without a Metropolis-Hastings correction can fail. The example here is due to Johnson [18] (personal communication). Suppose that we have a Gibbs sampler on  $\boldsymbol{\theta} = (\theta_1, \dots, \theta_m)$  with target distribution  $\pi \sim N_m(\mathbf{0}, \boldsymbol{\Sigma})$ .

Consider the following partially synchronous sampler with workers  $(w_1, \dots, w_m)$ , each of which updates one coordinate. Initialize arbitrary  $(\theta_1^0, \dots, \theta_m^0)$  and, in parallel, update the following:

$$w_1 : \text{update } \theta_1^1 \mid \theta_2^0, \dots, \theta_m^0 \quad \dots \quad w_m : \text{update } \theta_m^1 \mid \theta_1^0, \dots, \theta_{m-1}^0. \quad (22)$$

Now synchronize by writing to shared memory and then reading from it, and repeat indefinitely. This sampling scheme does not converge for all  $\boldsymbol{\Sigma}$  [5]. In particular, it can diverge if the precision matrix  $\boldsymbol{\Sigma}^{-1}$  is not diagonally dominant. Furthermore, even when it does converge, the sample covariance matrix of the output can be incorrect. We call this algorithm *Jacobi sampling*, because the mean vector at each update is an iteration of the Jacobi algorithm for solving linear systems [31] – for the corresponding linear system, diagonal dominance suffices to ensure stability of the iterations.

We analyze the following case:

$$\boldsymbol{\Sigma}^{-1} = \begin{bmatrix} 1.01 & 1 & \dots & 1 & 1 \\ 1 & 1.01 & \ddots & 1 & 1 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 1 & 1 & \ddots & 1.01 & 1 \\ 1 & 1 & \dots & 1 & 1.01 \end{bmatrix} \quad \boldsymbol{\Sigma} = \begin{bmatrix} 87.5 & -12.5 & \dots & -12.5 & -12.5 \\ -12.5 & 87.5 & \ddots & -12.5 & -12.5 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ -12.5 & -12.5 & \ddots & 87.5 & -12.5 \\ -12.5 & -12.5 & \dots & -12.5 & 87.5 \end{bmatrix}. \quad (23)$$

This is clearly a rather difficult target from the parallel sampling perspective due to strong dependence in the components of  $\boldsymbol{\theta}$ . We call the  $\boldsymbol{\Sigma}$  in (23) the Jacobi covariance matrix.

Jacobi Sampling with a target that has the covariance matrix (23) diverges to  $\infty$ . What goes wrong? Without the MH accept-reject step, the algorithm is a Noisy Monte Carlo [1] approximation to exact Asynchronous Gibbs: the MH acceptance probability is replaced by a biased estimator, namely 1. If this approximation is bad, the algorithm can fail.

For comparison, consider a correlated 8-dimensional Gaussian with mean zero, variance 1, and exponential covariance function  $\boldsymbol{\Sigma}_{ij} = \exp\{-\phi|i-j|\}$  with  $\phi$  set to 0.5. By comparing this target to the one specified by (23), we seek to provide some characterization of when the approximation of the MH acceptance probability by 1 is good, and therefore when the MH correction in exact Asynchronous Gibbs can safely be ignored.

To this end, we now examine a variation of Jacobi sampling that numerically converges to the correct mean and incorrect covariance matrix if the approximate algorithm is used. Suppose that there are 4 workers, each with 2 full conditionals assigned to them from our 8-dimensional Gaussian target. Each worker selects one of its full conditionals at random, performs a Gibbs step, and transmits the resulting draw to each other worker with probability 0.75. The other workers then perform a Metropolis-Hastings calculation and either accept or reject the transmitted value. As long as the MH correction step is performed, this algorithm's convergence is implied by our theorem in Section 6. If the MH step is ignored, the algorithm's convergence will depend on the target covariance matrix.



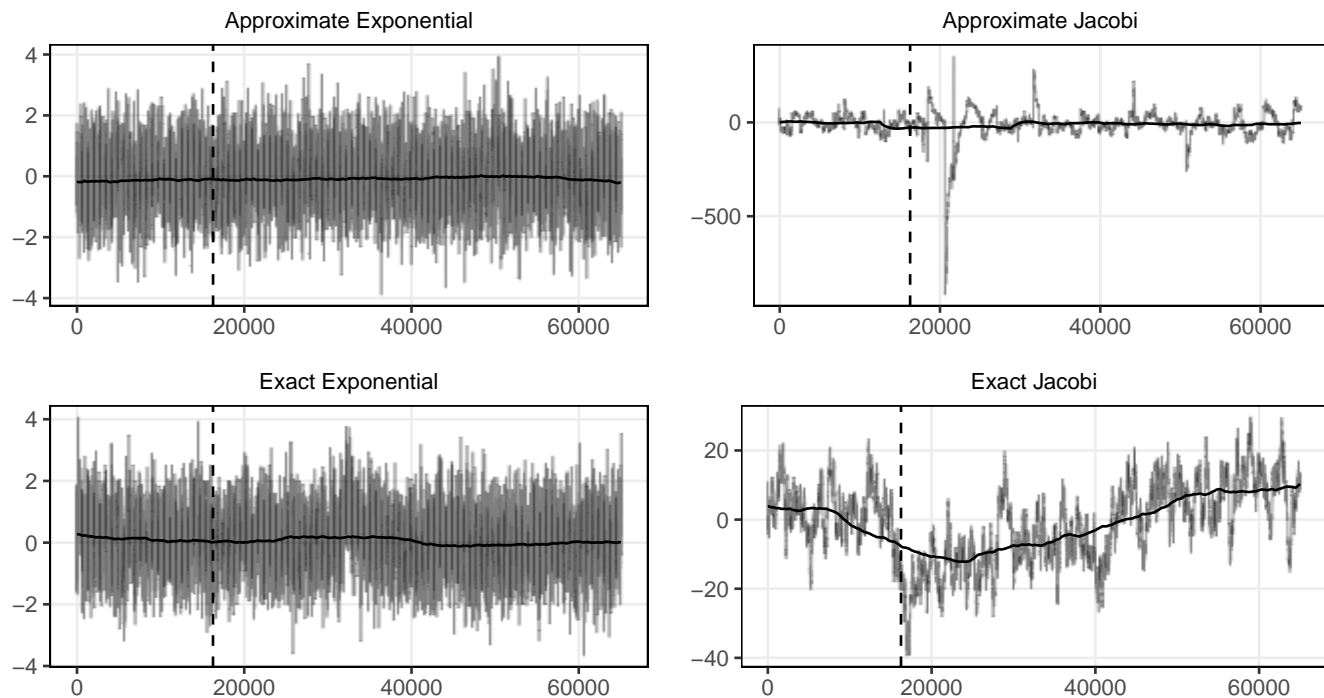


Figure 3: Diagnostic for the Jacobi sampling algorithm, exact and approximate variations, with exponential and Jacobi covariance matrices: approximate exponential (upper left), approximate Jacobi (upper right), exact exponential (lower left), exact Jacobi (lower right).

We implemented both the exact and approximate versions of this variation with the covariance matrix (23) on one machine with simulated parallelism. For comparison, we also ran the variation with the exponential covariance matrix. Diagnostic plots are given in Figure 3. Clearly, the algorithm does far better with the exponential covariance. The exact algorithm with Jacobi covariance matrix mixes poorly, but ends up yielding a Monte Carlo mean and covariance matrix that are not too far away from the correct answer. The approximate algorithm with Jacobi covariance matrix yields the correct mean, but vastly incorrect covariance matrix.

To further study the differences between the exact and approximate algorithm, we examined the distribution of the MH acceptance ratios in all four examples (Figure 4). In the case of the approximate algorithm this was accomplished by calculating and storing the MH probabilities and then ignoring them by accepting all updates. This distribution was concentrated around 1 for the approximate exponential case. It was substantially lower – bimodal near 0 and 1 – for the approximate Jacobi case that yielded the wrong answer. Interestingly, the MH ratio distributions were also different when comparing both exact algorithms to their approximate counterparts. We are not sure how to interpret this difference, other than to note that the exact and approximate algorithms may behave differently even in situations where the approximation is relatively good.

The intuition suggested by this example leads to the following diagnostic:

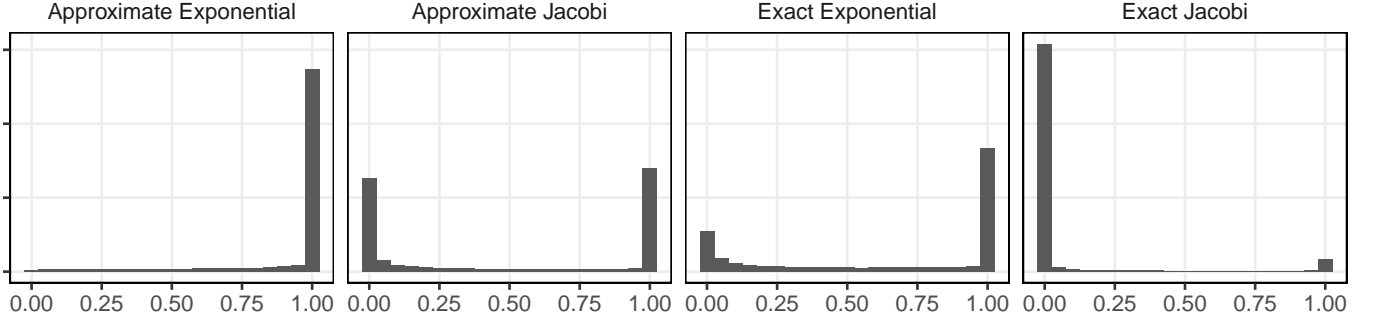


Figure 4: Distribution of the Metropolis-Hastings acceptance ratio for both approximate and exact Asynchronous Gibbs, with the algorithm described in Section 6.3, and exponential and Jacobi covariance matrices.

#### Diagnostic A.

Approximate Asynchronous Gibbs is reasonable if the distribution of the Metropolis-Hastings acceptance ratio in the approximate algorithm is concentrated around 1.

If the condition in Diagnostic A is satisfied, the behavior of the approximate algorithm will be similar to that of the exact algorithm in the posterior regions that it explores. This suggests that the bias introduced through the Noisy Monte Carlo [1] approximation should not be too large.

Further work is needed to formalize this intuition. In particular, additional results on Noisy Monte Carlo would grant additional understanding of how such approximations can affect the results of the algorithm. See Section 7 for additional discussion.

To conclude, we provide the following heuristic for describing problems in which the approximate algorithm is appropriate.

#### Heuristic B.

Asynchronous Gibbs without Metropolis-Hastings correction produces a good approximation to the exact algorithm if all of the following hold:

- (i) The target density  $\pi$  does not possess too much dependence between its coordinate components.
- (ii) The dimensionality of  $\pi$  is significantly larger than the number of workers.
- (iii) All transmitted variables are drawn via Gibbs steps.

We propose Heuristic B for the following reasons: (i) suggests that full conditional distributions in nearby posterior regions are similar, (ii) suggests that there is not too much movement happening at once, and (iii) suggests, given the previous two conditions, that the algorithm will consist of moves that are approximately Gibbs steps and hence should be accepted often.

Both Diagnostic A and Heuristic B are intuitive tools designed to help practitioners use approximate Asynchronous Gibbs in situations where it is likely to work well. We cannot at this time prove any theorems formalizing the intuition that we have provided. Future work is necessary to understand this aspect of the algorithm.

## 7 Discussion

Approximate Asynchronous Gibbs sampling appears to offer a way forward for Bayesian inference at scale in a large class of models (as noted previously, the exact algorithm will not scale as well as the approximate algorithm, because the number of Metropolis-Hastings evaluations that need to be performed will increase with the number of workers). In particular, models with a structure similar to the one found in the hierarchical mixed-effects regression example of Section 6.2 – in which each data point has its own latent variable – appear especially promising, because the dependence in the posterior between almost all dimensions, for instance two vectors  $\beta_i, \beta_j$  for  $i \neq j$ , is weak. Models with strong posterior dependence will likely remain difficult, because we expect poor mixing in that context. One way around this would be to tailor blocking of the Gibbs sampler to the specific problem at hand. For example, the performance of the Gaussian Process regression in Section 6.1 could be further improved by considering an overlapping block scheme, such as the *Additive Schwartz* method [31].

Other models, such as those involving high-dimensional (for example, of order 10,000 or more) Inverse Wishart priors, remain intractable because they unavoidably require the construction of matrices that are too big to invert quickly, or perhaps even too big to fit into a single computer’s memory. Similarly, models that depend strongly on each individual data point will not perform well in this context, as some amount of data-level redundancy is needed to make the scheme robust against temporary worker failures. Potential solutions to these problems may not arise wholly from within the realm of statistical computing, but may also rely on new insights into the Bayesian modeling of large data sets.

Finally, models involving excessive *reduce* operations over large portions of the data also remain intractable, because these computations massively slow down the speed at which individual steps of Asynchronous Gibbs proceed. This difficulty can be mitigated through caching, but this may result in unacceptably slow mixing.

The theory of Asynchronous Gibbs sampling can be further expanded. While we focused on convergence, it would be useful to quantify the degree to which a lack of synchronicity affects the performance of the algorithm. Asynchronous Gibbs produces a higher order Markov chain, the direct analysis of which is non-trivial. However, we believe that a more detailed understanding of the interplay between the convergence behavior of the chain and the Markov chain’s order will be a useful step toward developing “partially asynchronous” MCMC methods, which may mix better or possess other useful properties, and could potentially use asynchronous steps to hide latency during the global *reduce* operations required for synchronization. This would mirror recent advances in massively parallel iterative algorithms for solving linear systems [14].

Further research in Noisy Monte Carlo [1] is needed to understand the approximate algorithm. By replacing the Metropolis-Hastings ratio with a biased estimator (namely 1), we are able to get significantly better scaling, but this introduces some bias into our answer. A variety of results would help in understanding this bias. It is also unclear whether properties such as geometric ergodicity are inherited by the synchronous parallel chain  $H$  from the underlying chain  $P$ . Further work is needed to understand the approximate algorithm from these perspectives.

Asynchronous Gibbs sampling is nontrivial to implement. It is a fully parallel method that in our implementation mixes thread-based and actor-based parallelism, which is widely considered in the parallel computation literature to be a distinctly bad idea due to the coding complexity involved,

including increased potential for *race conditions* and other nondeterministic errors that are difficult to debug. All of the data structures used need to be capable of functioning correctly in parallel. We highly recommend *Scala* as the programming language of choice, because its design makes it much more compatible with parallel programming than most other languages.

Implementation is also specific to both the problem being solved and the hardware used – in particular, it is necessary to decide how to divide the workload among all of the workers. We found that different choices produced widely different mixing efficiencies – in the extreme, one worker can bottleneck the entire algorithm if it is sampling, at too slow a rate, a dimension upon which all other workers depend. Similar issues can occur with respect to network traffic control: if one worker is producing output too fast, it can flood the network, preventing other workers from communicating with each other. This is not solely an issue in complex problems – at one point in time (due to an incorrect *Akka* configuration) this difficulty manifested itself in a simple problem involving an 8-dimensional Gaussian: because the variables can be sampled quickly, a large number of them were produced, resulting in significant network traffic in spite of the problem’s small size. Thus care was required to properly tune the algorithm and ensure that it converged in the problems we studied.

Our current implementation is nowhere near optimal. *Akka* is designed for large-scale distributed applications rather than high-performance computing. This makes for convenient development, but does not give us the kind of low-level hardware control available in a framework such as *MPI*. Our cluster also was selected for convenience rather than performance – indeed, the machines of which it was comprised were physically located in data centers in three different US states. This is an extremely high latency environment from a high-performance computing perspective. While this illustrates our algorithm’s robustness, it also means that we do not know how it will perform in a traditional *MPI*-style scientific computing environment on a standard supercomputer.

These challenges are common to any parallel computation scheme, where fully generic solutions are difficult. Instead, we have focused on building Asynchronous Gibbs for solving a common class of Big-Data Bayesian problems, for which the number of latent variables grows with the number of data points. In doing this, we traded off some sequential efficiency for the promise of better parallel behavior, and we find that the initial results are promising.

Asynchronous computing is almost completely unexplored for Bayesian computation. In our view, Asynchronous Gibbs Sampling makes a strong argument that this should no longer be the case.

## Acknowledgments

We are grateful to Chris Severs, Joseph Beynon, and Matt Gardner for many useful discussions and much help with the coding. DS (and perhaps to a lesser extent AT and DD) would like to thank Christian Robert for sending him an early version of this paper that allowed him, during an otherwise uneventful trip to buy trousers, to remember a strategy of proof for asynchronous algorithms. We are grateful to Matt Johnson for sending us the *Jacobi Sampling* example and thereby demonstrating that approximate Asynchronous Gibbs is not exact, the subsequent analysis of which made much of this work possible. We thank Christopher De Sa for sending us a different counterexample to exactness of the approximate algorithm on  $\{0, 1\}^3$ . We are grateful to Cyril Chimisov for pointing out a subtle mathematical error in our original proof. We thank Herbie Lee for his ideas, which simplified the arguments used in our proof. We are grateful to Gareth

Roberts, Murray Pollock, Radford Neal, Daniele Venturi, Rajarshi Guhaniyogi, Tatiana Xifara, and Torsten Erhardt for interesting and useful discussions. We also thank eBay, Inc. for providing the computational resources used for running our examples. Membership on this list does not imply agreement with the ideas given here, nor are any of these people responsible for any errors that may be present.

## References

- [1] P. Alquier, N. Friel, R. Everitt, and A. Boland. Noisy Monte Carlo: Convergence of Markov chains with approximate transition kernels. *Statistics and Computing*, 26(1-2):29–47, 2016 (cited on pages 4, 16, 18, 19).
- [2] A. Banerjee, D. B. Dunson, and S. T. Tokdar. Efficient Gaussian process regression for large datasets. *Biometrika*:75–89, 2012 (cited on page 12).
- [3] G. M. Baudet. Asynchronous iterative methods for multiprocessors. *Journal of the Association for Computing Machinery*, 25(2):226–244, 1978 (cited on pages 5, 8).
- [4] D. P. Bertsekas. Distributed asynchronous computation of fixed points. *Mathematical Programming*, 27(1):107–120, 1983 (cited on pages 5, 8).
- [5] D. P. Bertsekas and J. N. Tsitsiklis. *Parallel and distributed computation: numerical methods*. Prentice-Hall, 1989 (cited on page 16).
- [6] R. T. Cox. Probability, frequency and reasonable expectation. *American Journal of Physics*, 14(1):1–13, 1946 (cited on page 1).
- [7] C. De Sa, K. Olukotun, and C. Ré. Ensuring rapid mixing and low bias for asynchronous Gibbs sampling. *arXiv:1602.07415*, 2016 (cited on pages 3, 4).
- [8] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society, Series B (Statistical Methodology)*:1–38, 1977 (cited on page 8).
- [9] M. Dow. Explicit inverses of Toeplitz and associated matrices. *ANZIAM Journal*, 44:185–215, 2008 (cited on page 12).
- [10] A. Frommer and D. B. Szyld. On asynchronous iterations. *Journal of Computational and Applied Mathematics*, 123(1):201–216, 2000 (cited on pages 7, 8).
- [11] R. Furrer, M. G. Genton, and D. Nychka. Covariance tapering for interpolation of large spatial datasets. *Journal of Computational and Graphical Statistics*, 15(3), 2006 (cited on page 12).
- [12] A. E. Gelfand and A. F. M. Smith. Sampling-based approaches to calculating marginal densities. *Journal of the American Statistical Association*, 85(410):398–409, 1990 (cited on page 2).
- [13] S. Geman and D. Geman. Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, (6):721–741, 1984 (cited on page 2).
- [14] P. Ghysels and W. Vanroose. Hiding global synchronization latency in the preconditioned Conjugate Gradient algorithm. *Parallel Computing*, 40(7):224–238, 2014 (cited on page 19).
- [15] W. K. Hastings. Monte Carlo sampling methods using Markov chains and their applications. *Biometrika*, 57(1):97–109, 1970 (cited on pages 1, 2).

- [16] G. Hu and R. F. O’Connell. Analytical inversion of symmetric tridiagonal matrices. *Journal of Physics A: Mathematical and General*, 29(7):1511–1513, 1996 (cited on page 12).
- [17] A. Ihler and D. Newman. Understanding errors in approximate distributed Latent Dirichlet Allocation. *IEEE Transactions on Knowledge and Data Engineering*, 24(5):952–960, 2012 (cited on page 3).
- [18] M. Johnson. Jacobi sampling, 2015. Personal Communication (cited on page 16).
- [19] M. Johnson, J. Saunderson, and A. Willsky. Analyzing Hogwild parallel Gaussian Gibbs sampling. In *Advances in Neural Information Processing Systems*, pages 2715–2723, 2013 (cited on page 3).
- [20] A. Kottas. Bayesian inference for Gaussian process nonparametric regression. Personal Communication, 2014 (cited on page 11).
- [21] J. S. Liu, W. H. Wong, and A. Kong. Covariance structure and convergence rate of the Gibbs sampler with various scans. *Journal of the Royal Statistical Society, Series B (Statistical Methodology)*:157–169, 1995 (cited on page 2).
- [22] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller. Equation of state calculations by fast computing machines. *The Journal of Chemical Physics*, 21(6):1087–1092, 1953 (cited on pages 1–3).
- [23] S. P. Meyn and R. L. Tweedie. *Markov Chains and Stochastic Stability*. Springer-Verlag, 1993 (cited on page 9).
- [24] R. M. Neal. Monte Carlo implementation of Gaussian process models for Bayesian regression and classification. *arXiv:physics/9701026*, 1997 (cited on page 11).
- [25] R. M. Neal. Slice sampling. *The Annals of Statistics*, 31(3):705–741, 2003 (cited on page 3).
- [26] W. Neiswanger, C. Wang, and E. Xing. Asymptotically exact, embarrassingly parallel MCMC. *arXiv:1311.4780*, 2013 (cited on page 2).
- [27] F. Niu, B. Recht, C. Re, and S. Wright. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In *Advances in Neural Information Processing Systems*, pages 693–701, 2011 (cited on page 8).
- [28] G. O. Roberts and J. S. Rosenthal. Surprising convergence properties of some simple Gibbs samplers under various scans. 2015 (cited on page 2).
- [29] G. O. Roberts and R. L. Tweedie. Exponential convergence of Langevin distributions and their discrete approximations. *Bernoulli*:341–363, 1996 (cited on page 3).
- [30] G. Roberts and J. Rosenthal. Coupling and ergodicity of adaptive MCMC. *Journal of Applied Probability*, 44:458–475, 2007 (cited on page 6).
- [31] Y. Saad. *Iterative methods for sparse linear systems*. SIAM, 2nd edition, 2003 (cited on pages 16, 19).
- [32] S. L. Scott, A. W. Blocker, F. V. Bonassi, H. A. Chipman, E. I. George, and R. E. McCulloch. Bayes and Big Data: The Consensus Monte Carlo algorithm. In *Bayes 250*, volume 16, 2013 (cited on page 2).
- [33] S. Srivastava, C. Li, and D. B. Dunson. Scalable Bayes via Barycenter in Wasserstein Space. *arXiv:1508.05880*, 2015 (cited on page 2).

- [34] M. L. Stein, Z. Chi, and L. J. Welty. Approximating likelihoods for large spatial data sets. *Journal of the Royal Statistical Society, Series B (Statistical Methodology)*, 66(2):275–296, 2004 (cited on page 12).
- [35] R. H. Swendsen and J.-S. Wang. Replica Monte Carlo simulation of spin-glasses. *Physical Review Letters*, 57(21):2607–2609, 1986 (cited on page 5).
- [36] A. Terenin and D. Draper. Rigorizing and Extending the Cox-Jaynes Derivation of Probability: Implications for Statistical Practice. *arXiv:1507.06597*, 2015. URL: <http://arxiv.org/abs/1507.06597> (cited on page 1).
- [37] V. v. Brzeski, M. Taddy, and D. Draper. Causal Inference in Repeated Observational Studies: A Case Study of eBay Product Releases. *arXiv:1509.03940*, 2015 (cited on page 13).